# GasMASk Annotation-based Code Generator as an Embedded Domain-Specific Language in Collaborative Multi-Agent Systems

Orçun Oruç, Uwe Aßmann

TU Dresden, Software Technology Group, Nöthnitzer Straße 46, 01187, Dresden

**Abstract.** Multi-agent systems have evolved with their application framework, analysis approaches and design complexities over the past few decades. In order for programming software agents, you need to handle analysis, design and implementation together; furthermore, privacy and trust should be integrated externally into the agent-oriented applications. Current frameworks do not consider privacy, trust, and data integrity in agent applications.
In this study, we propose a domain-specific language that can be used for agent behaviors that consist of roles, organizational entity, goals through annotation processing with templates. Smart contracts can be generated to decrease the time for deployment and development stages. Templates and annotations are popular techniques to reduce boilerplate codebases from agent-oriented programming. These techniques can also be used for model-driven software engineering. This study will take the software agent development as a whole with analysis, design, and development with embedded domain-specific language development in terms of smart contract applications. Furthermore, we would like to refer to methodology, results of the research, and case study to enlighten readers in a better way. Finally, we summarize findings and highlight the main research points by inferencing in the conclusion section.

**Keywords:** Software agents, Domain-specific languages, Blockchain technology, Smart contracts, Role-based programming languages, Code Generation, Template-based code generation

## 1 Introduction

Agent-oriented programming (AOP) can be considered as a subject of object-oriented programming by showing the state of an object with human-like features such as belief, desire, intentions, and goals. Moreover, an agent should be in the interaction with other agents, in this way, agents are able to play roles as human-being does. AOP specializes the object-oriented programming methodology by fixing state and modules, which are called agents, to consist of features that come from agent behaviors . Besides, an agent can handle the message passing between other agents internally.

Agent-based systems have changed their characteristics over the past few decades aspect of design, analysis, and implementation. Although one can find out the dif-

DOI : 10.5121/ijsptm.2021.10201

ficulty of exact definition in terms of the multi-agent system, multi-agent systems are used broadly in the application areas such as supply chain management, distributed systems, smart grids, robotic motion planning. Multi-agent systems are strongly dependent on contexts and roles. Each agent plays a role and it has a minimal set of attributes that represents the environment. Moreover, agents should have behaviors that are, in essence, related to implementing deterministic or non-deterministic behaviors of an agent that operate a role that can be in sequential order, cyclic order, or parallel order.

Software agents have some mandatory features in order to complete tasks that have been given by developers. Agent must percept the surrounding environment through information received from the environment by sensors. And then an event should be created by an agent to create knowledge about the environment. This has been called beliefs, which provides some aspect of the agent's knowledge about the environment [1]. After defining a concrete goal for an agent, events are often extracted from percepts and triggered by a new action [2].

Agents must be in a relationship with external trusted parties to provide privacy and consistency in multi-agent systems. However, the trust was mostly provided by different logic interpretations and cumbersome ontological definitions in multi-agent systems. Blockchain technology offers a credible and private data pool that can be used with programmable contracts (smart contracts) as a shared database. As we solved the credibility problem with blockchain technology, we can enforce the data layer security, which is vital for data-driven multi-agent system communication, by implementing smart contracts on the application layer of the blockchain protocol. A smart contract is a piece of code that is stored on a blockchain by triggering coin-based transactions with saved data and which reads and writes data in a blockchain database [1]. In addition, smart contracts can ensure the testability of role features, secure transactions within the blockchain database, and separation of the business logic (model) and application logic (system architecture).

The research paper is structured as follows. First, we will discuss the importance and short history regarding the topic in the Background 2. In the Methodology and Contributions 1.1 chapter, we will explain contributions that can occur at the end of this research and emphasize our objectives, problem statement, and research questions. Secondly, we will list some limitations 1.2, which we might encounter during this research. In the chapter 3, we introduce previous studies and surveys that have been conducted before. Later, we will give the details of the implementation in the chapter 4. In the chapter called Case Study 5, we will introduce a sample concept, which we would like to implement with our language. In chapter 6 and 7, we will list our findings and tasks that should be taken into consideration for further research.

[1] `https://www.coindesk.com/three-smart-contract-misconceptions`

### 1.1 Methodology and Contributions

First of all, we have two research questions. To conclude up regarding the research problem, we have defined research questions (RQ) as below:

- RQ1: Can a domain-specific language that comprises the main features of agent communication language, agent framework, and smart contract language be created?
- RQ2: Can templates be integrated into annotation processors in terms of roles, goals, and compartments in the domain-specific language?

**Goal:**

The main goal of this research is to implement a domain-specific language to demonstrate template samples and code generation for smart contracts.

**Objectives:**

- Implementing templates to create and delete agents in one of the selected frameworks.
- Defining annotations for stateful smart contract language in order for the process of code generation.
- At the implementation phase, we will define some events in smart contracts to log to the blockchain networks and specify agents' basic features to be alive in an agent management system.

Expected contributions will be in the following items:

- We will have a new embedded domain-specific language that takes as a whole privacy, security, data integrity and role-orientation in multi-agent systems.
- We will create a tool for stateful contract development in the blockchain database.
- We will have agent templates to reduce boilerplate codebase and increase the learning curve for agent-oriented applications in a practical framework.

### 1.2 Limitations

In this section, we will list our limitations regarding the process of writing the thesis.

- We will present code templates for agent-oriented software engineering and discuss with the metaprogramming through annotation processing.
- We will focus on the existing meta-model such as CROM for specifying communicative entity types. In the context of CROM research, we will follow the guideline regarding roles and compartments that have been specified before.
- We will develop an application based on a stateful smart contract language such as Solidity the following design-by-contract approach that interacts with different agents in the context of role-oriented programming.

– This research is limited to the KQML and FIPA agent communication languages and it does not comprise stateless blockchain language. Due to the nature of the stateless blockchain language, it does not purely suitable for the object-oriented approach.
– As for ontological representation, semantic heterogeneity between agent-oriented frameworks will not be taken into consideration. So we basically will handle existing ontologies and will not advance to ontology engineering.

## 2 Background

Code generation allows you to produce program code that is only strongly typed, and yet can be easily reconfigured codebase when the source model is changed [2]. Principally, code generators are metaprograms translating a regular tree to a sentence of a context-free language [3]. When the code generator translates the abstract syntax tree into a context-free language, it uses a model, which is the definition language of the agent, and view(result) in order for realizing the operation. An abstract syntax tree is a tree representation of the syntactic structure that has been represented in a programming language. In this study, we will write a code-generator in an ad-hoc fashion, which is going to be written for agent-oriented software applications. The code generation process is a projection of the input data model into to output codebase. By defining language vocabulary in our embedded-domain modeling language, we can easily generate an agent-oriented codebase for various practical frameworks.

### 2.1 Agent-Oriented Programming

An agent is anything that can be viewed as perceiving its environment through input and output appliances and acting upon that environment
[4].To understand an environment, agent needs to register the state of history regarding the environment. Agents have been designed to better understand the complexity of software systems. Agent-to-Agent and Agent-to-Environment interactions are an important part of agent-oriented programming because the state of passive objects is supposed to have interacted in a dynamic environment.

The model of goals, roles, and organizations should be designed at the system level. In the practical frameworks, this can be done by metamodels or metaprogramming with an embedded domain-specific language. Both of the methods can generate agent-oriented code with smart contracts, but changes should be integrated into the metamodels. Model and view should be strictly separated in both approaches and the metamodel approach takes more time for a steep learning curve.

[2] Microsoft Code-by-using-templates

Organizational participants are one of the most important features in the agent-oriented world. For instance, the car dealership, the bank, the school, and the insurance company are all examples of organizational participants [5]. Agent orientation aims at pursuing a software paradigm. The main research point is about whether the agent-oriented can ensure a higher quality than its predecessor in terms of autonomy, reactivity, proactivity, and cooperative features in a particular domain.

In practice, roles, organizations, and goals are defined by messages that contain envelope, payload, message, and content [**?**]. Agent platforms are physical platforms in which agents are deployed and organized. Agent platforms need to register to directory facilitator in order to find other agent services in the network [**?**]. An agent can modify its agent description by sending requests to the directory facilitator. Agents are deleted or created by agent management systems, which are one of the core components in practical agent frameworks. Lastly, organizational and role-based entities are described in messages in compliance with FIPA and KQML through agent messaging transport services.

### 2.2 Role-oriented Programming

In an object-oriented application, objects are in a relationship between other objects and they are sharing their features of relationships through constraints, dependency, and invariants. Even though there is no common understanding about roles in a programming language, role-oriented programming basically looks for an answer to whether or not an object can collaborate with defined roles among other objects in the application lifetime.

Roles encapsulate the dynamic behavior of players, which are the core objects in the context, and dynamically adapt the players' behavior [6]. Compartments are container that holds multiple roles or their collaboration features. Roles can make the design of multi-agent systems easier by implementing the composition of role attributes, role invariants, role methods, and binding interfaces. The difference between roles and objects is whether or not the roles can move hosts that exist in the environment[7]. Role-based software agents are related to context-aware multi-agent systems. Context is any information that is accessible to the program, where an entity is a person, place, or another agent that is considered relevant to the determination of behavioral variations [8]. Agents can dynamically collaborate with roles, create coalitions of trusted partners as an effective mechanism to communicate with service requestors, find services requested by them, and determine trusted services and provide services to the applicants without violating the privacy of the predefined environment [9]

Compartments belong to the research of the Compartment Role Object Model (CROM) that establishes subtypes of natural types and relationship types among combined roles [10]

### 2.3 Smart Contracts

Smart contracts are a piece of code that can be triggered by blockchain network through transactions. More specifically, a smart contract is a secure and unstoppable computer representing an agreement that is automatically executable and enforceable [11]. A transaction is a signed data package to transfer value, which can be coinbase value, from an account to another account or to a contract, invoke a method of a contract, or deploy a new contract [11]. Principally, an organization has an identity and its own purpose [11].

Smart contracts are deterministic applications, which means that the applications produce the same outputs every time they are carried out. They also have some preconditions and postconditions that focus on the condition logic of applications rather than syntax or algorithms in practice.

Blockchain networks behave in a distributed and decentralized manner, which provides immutable, permissionless or permissioned solutions in a private or public environment. Consensus is the backbone mechanism of a blockchain solution that is comprising of various types of them such as proof-of-work, proof-of-stake, proof-of-authority. The selection of types of consensus should be defined by blockchain technology, so the technology choice is one of the important parameters for smart contract applications. For instance, Corda applications do not provide certain determinism just like the Ethereum technology because developers have designed the Corda blockchain network as a distributed ledger without designing a particular virtual machine environment. In order to ensure deterministic behaviors in the blockchain networks, they must imply honest agreement, proper termination, the validity of nodes, and fault-tolerant.

One can transfer value with a fixed or dynamic price in the blockchain network, which is called on-chain transactions. For deterministic value transfer, on-chain transactions must be executed in the blockchain virtual machine. Stateful or stateful transactions can be made in either declarative programming languages or imperative programming languages. Agents who want to join the network should take permission from the network administrator. Hence, we can restrict the access for any external participants who want to connect with the internal blockchain network that is called permissioned blockchain. This provides high throughput and low latency in the blockchain network. However, the nature of blockchain is always permissionless because all nodes can participate in the network to synchronize their internal block data structures.

**Dynamic Smart Contracts** For permissionless and permissioned blockchain, once a smart contract is deployed, the smart contract remains immutable [12]. A smart contract can execute asset-value transfers in the blockchain network or execute an external contract by passing its address. Thus, a smart contract can

trigger another smart contract or itself with different attributes. This is the point that we can create dynamic smart contracts.

Dynamic smart contracts deal with the maintainability of immutable smart contracts in order to change their behaviors at runtime. Reconfiguration of a smart contract that is called by multiple contracts might have seen effective to implement; however, deployment overload can affect the performance of the blockchain network. Dynamic smart contracts are not implemented in this study; nevertheless, we will consider the behavior changes using parameter passing from external contracts to realize partly dynamic states for agents in the blockchain network.

### 2.4 Domain-Specific Languages for Metaprogramming

Domain-Specific Languages (DSLs) have a small set of features and they are mostly optimized for a specific domain such as health, finance, or manufacturing. By optimizing for a specific domain, a DSL can be easily maintained and built upon a general-purpose language without defining deeply the underlying concept of language design. In this study, we have chosen a general-purpose language called Java, which provides Pluggable Annotation Processing API that deals with advanced customized annotations at compile or runtime levels. In this study, we basically generate an embedded domain-specific language that generates a piece of code in a general-purpose language and a stateful smart contract language. At the first stage of the research, applications will be generated by the embedded domain-specific language. Later, we will focus on the feature of language whether or not it changes the behavior of existing programs.

## 3 State of the Art

ALAADIN is one of the oldest metamodels [13] to define models of organizations for agents and this model defines a very simple description of coordination and negation schema [13]. The authors determine that the role is an abstract representation of an agent or service function within a group. Groups are a set of features that behave as an atomic entity so that an agent dynamically joins, creates, or leaves groups [14].

When we focus on behavioral roles for agent interaction, (Cabri et. al. 2003) proposed that an agent system defines a role as a set of capabilities and expected behaviors. BRAIN is an approach that covers a role-based interaction model, where agents' interactions and behaviors are embedded in roles [14]. Moreover, they achieved and advise to realize agent-oriented features, separation of concerns, and reuse of solutions [15]. To describe agents semantically, they defined a language called XRole that exploits built-up definitions of roles. These definitions consist of name, description, addresses, role description, and contents of the agents with relational features such as *MinOccurs* and *MaxOccurs*. RoleSystem is an interaction

infrastructure that implements the model of BRAIN [16]. Roles defined by XRole can be read by humans as well as by agents and tools [16]. The RoleSystem provides two main components which are: *reqRegistration*, to register an agent in the system with a specified role; *searchForRoleAgent*, to search for agents playing a given role between agents and server agents [16].

The planning capability of multi-agent systems is one of the key features that the blockchain should take care of it. After assigning roles, plan execution of the multi-agent systems should complete distributed ordering actions. To do so, a smart contract can be used which are essentially collections of distributed code and data representing some business logic that works with the blockchain distributed consensus protocols [17]. The main idea of this paper is to coordinate the steps of multi-agents through the smart contracts aspect of distributed plan execution. In this plan execution, multiple smart contracts can be used such as oracle contract, which is allowing to exploits data in the off-chain storage, or contract of preconditions and postconditions to provide the design-by-contract pattern.

Gaia is one of the methodologies at the design and analysis phases in multi-agent systems. The main goal of this methodology is to model multi-agent systems for an organization where different roles interact [14]. The Gaia methodology defines the features of roles as below:

- *Responsibilities*: They specify the functionalities of agents that play roles.
- *Permissions*: They are a set of rights associated with roles in which agents play.
- *Activities*: Internal computation of an agent. This does not take into consideration the relationship between agents.
- *Protocols*: This is related to interaction roles indicating agent-to-agent communication.

The role-based evolutionary programming (RoleEP) presents cooperative mobile agents to collaborate in achieving a common goal [14]. The authors of RoleEP state that an object becomes an agent by binding itself to a role that is defined in a dynamic environment [7]. The authors have defined the basic concept as below [7].

- *Environment:* An environment is composed of environment attributes, methods of environment, and roles.
- *Roles:* A role, which can move between hosts that exist in an environment, contains role attributes, role methods, and binding interfaces.
- *Objects:* An object, which cannot move between hosts, is composed of attributes and methods.
- *Agents:* An object that binds itself with some roles and acquires traveling/collaboration functions is named as Agents.
- *Binding Interface:* A binding interface, which looks like an abstract method interface, is used when an object binds itself with a role.

Implementation of a domain-specific language may have metamodel design paths at the level of M1 (User Model), M2 (Unified Modeling Language), M3 (Meta Object Facility). For instance, AgentDSL is a domain-specific language for cross-cutting concerns for agents, which is supporting aspect-oriented programming, and non-crosscutting concerns [18]. The authors of AgentDSL defines a code generator that maps abstractions.

## 4 Implementation

Principally, the selection of the general-purpose language is an important parameter regarding the implementation of code generators because some languages give native-in support for custom annotations. For instance, Java programming language supports Annotation Processor which is equivalent to a plug-in of the compiler.

### 4.1 Algorithm of the Code Generation with Metaprogramming

**Algorithm 1** Code Generation with Metaprogramming

```
1: procedure codeGeneration()(a, b)
2:     while RetentionPolicy.SOURCE and ElementType.METHOD do  ▷ Runtime or Compile
   Time ?
3:         Set <?extendsElement > elements ← roundEnvironment.getElementsAnnotatedWith(classname)
4:         for Element element : elements do                          ▷ scan for elements
5:             AgentImplement autoImplement = element.getAnnotation(AgentGenerator.class);
   ▷ generate unique ID
6:         end for
7:         error =!checkIdValidity(AgentGenerator.generate(), element);
8:         if !error then
9:             uniqueIdCheckList.add(AgentGenerator.generate());
10:        else
11:            throwsanexception
12:        end if
13:    end while
14: end procedure
```

The afore-mentioned algorithm can be used with any general-purpose language even though it has been mentioned for Java language. More or less, the annotation process follows steps that have already been explained to readers. The most important point about either the RetentionPolicy should assign particular configurations at runtime or generate codebase at the compile time. The developer should decide the behavior before it was developed in agent-oriented frameworks.

### 4.2 Template-based Code Generation

Apache Velocity[3] is a template project that has been provided by Apache Foundation. Templates can take inputs internally or externally through general-purpose language. One of the disadvantages regarding templates has lack of validation steps at the compile time. Nevertheless, this issue can be solved by post compile-time validation. Generated source code can be integrated with compiler plugin in order to check data type validation. The following code snippet has been demonstrated as below:

```
package ${packagename};
#set($classkeyword=".class")
#set($classwithString=".class.toString()")
#set($dfDescription="DFDescription")
#set($agentID = ".setName(getAID());")
#set($addServices=".addServices(serviceDescription);")

import jade.core.Agent;

public class ${entity} extends Agents {
  private static final long serialVersionUID = 1L;

  //------------------------------------------
  // AGENT SPECIFIC FUNCTIONS
  //------------------------------------------

  public void setup() {

    addBehaviour(getAgentBehaviour(this, $entity$classkeyword};
    DFAgentDescription $entity$dfDescription = new DFDescription();
    $entity$dfDescription$agentID
    ServiceDescription serviceDescription = new ServiceDescription();
    serviceDescription.setType($entity$classwithString);
    serviceDescription.setName($entity$classkeyword);
    $entity$dfDescription$addServices
    try {
      DFService.register(this, $entity$dfDescription);
    } catch(Exception ex)
    {
      ex.printStackTrace();
    }
  }

  protected void takeDown() {
    doDelete();
  }
}
```

[3] https://velocity.apache.org/

After compiling the program from a template, a validation step must be done by an external application programmable interface (API). Most of the programming languages provide the Compiler API that might generate diagnostics during compilation such as error messages, warning-debug-info level logs [4]. An embedded compiler plugin is a better way to do it; however, external compiler API can test the application between multiple versions and it provides plenty of benefits to the agent application programmer. In the following code snippet, the codebase has been taken from the JThereum Project [5].

The following lines show us a generated codebase from a stateful smart contract language.

```
*/
pragma solidity ^0.5.9;
contract SimpleEventDemo
{
  function emitEvents(string memory str1, string memory str2) public
  {
    emit SimpleEvent(str1, str2, false);
    emit SimpleEvent(str2, str1, false);
  }
  event SimpleEvent(string str1, string str2, bool switched);


}
```

**Listing 1.1.** Generated Solidity Stateful Contract

[H]

```
public class App implements ContractProxyHelper
{

        public void emitACoupleOfEvents(String s1, String s2) {
        ContractStaticImports.emitEvent(new SimpleEvent(s1, s2, false
    ));
        ContractStaticImports.emitEvent(new SimpleEvent(s2, s1, false
    ));
      }

      @EventClass
      static class SimpleEvent extends EventClassHelper {
        final String s1;

        final String s2;

        final boolean switched;

        public SimpleEvent(String s1, String s2, boolean switched) {
```

[4] https://docs.oracle.com/javase/8/docs/api/javax/tools/JavaCompiler.html
[5] https://jthereum.com/

```
18            this.s1 = s1;
19            this.s2 = s2;
20            this.switched = switched;
21          }
22        }
```

**Listing 1.2.** Java Contract Generator

### 4.3 Metaprogramming

Metaprogramming is the ability to write programs that generate source code defined by developers [19]. Beyond that in some programming languages such as Groovy and Ruby, one can change the behavior of existing methods in order for realizing dynamic metaprogramming. For instance, Java offers two types of static metaprogramming such as annotation processing and aspect-oriented programming to generate codebase; however, dynamic metaprogramming can be achieved by external third-party libraries.

A meta-object protocol (MOP) represents the semantic of the extendibility that resides in a programming language. The behavior of the program is determined by the MOP, including the aspect of the program in runtime and compile-time [19].

Every aspect of a program's mapping down onto the lower level substrate that is controlled by objects in compile or runtime [6]. These objects are called meta-objects, because the primary duty of these objects is to map the program, not to represent the predefined domain. For instance, the java.lang.reflect package in Java, one can create a self-configuring system with pluggable annotations and templates.

Principally, metaprogramming is divided into two main parts, which are runtime metaprogramming and compile-time metaprogramming. Compile-time metaprogramming lets you organize generated codebase at compile time. You can perform syntactic transformations and code optimization level to interact with the compiler during the code generation phase [19].

## 5 Case Study

Manufacturing scheduling is the process of assignment of timing for order, manufacturing, and delivery. So we should provide a good quality per unit and the number of units should be maximized per slot in the production line. At the same time, we need to minimize the waste of resource requirements and potential failures. Basically, the next state of smart contracts in a blockchain environment is completely determined by the current state.

We can create role types, natural types and compartment types in order for implementing a sample case for GasMASk like in the following figure 1. We would like

[6] https://wiki.c2.com/?MetaObjectProtocol

to implement a supply chain modeling by realizing data structures in role-oriented programming in smart contracts. Smart contracts will take the data structure and evaluate them in object-oriented programming with preconditions and postconditions as the unit testing framework does.

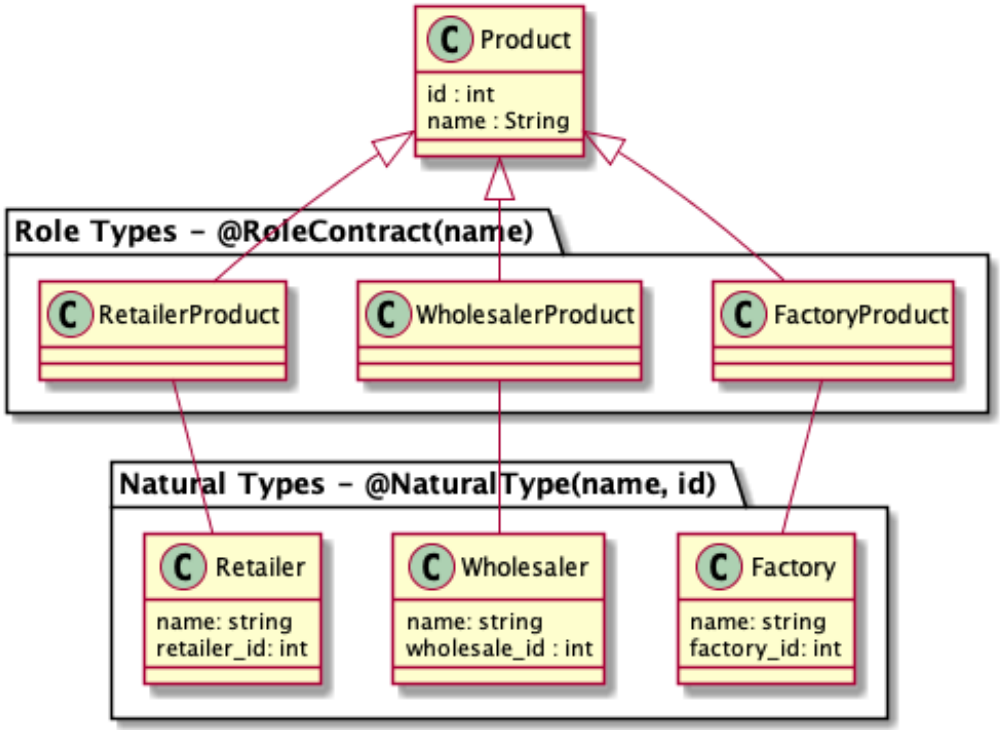


**Fig. 1.** Example of the role types

A product type is created for a manufacturing simulation to simulate between retailer, wholesaler, and manufacturing company. When we defined type of data structures for roles, they will be create through annotation processing by deploying in a blockchain network. All of the participants will see events, transactions and values in the blockchain network and the type of blockchain network will be permissioned blockchain to provide security and trust. RetailerProduct, WholeslarProduct, and FactoryProduct are the abstract types that are implemented from Product type. These types will be used for agents' behavior. Natural types in Figure 1 are going to be in Solidity language so that one can use them for verification and immutable behaviors.

## 6 Conclusion

The paper addressed the challenge of compelling trust and security in multi-agent systems and their role-oriented features by realizing smart contracts regarding blockchain technology (BCT). The main purpose of the research is to give a new approach to the intersection between smart contract programming, role-oriented programming, and agent-oriented programming. The course of findings among various research areas guides us to design a domain-specific language to contribute to the multi-agent system area.

- There is no common understanding in terms of multi-agent system methodology, analysis, design, or implementation. This increases the complexity of the research in the multi-agent system area.
- The limited number of domain-driven agent-oriented languages have been provided so that one can notice that multi-agent system research is most likely conceptual and it does not provide prototype and result-evaluated research.
- Synthesized metamodeling from scratch in different research areas can be ambiguous; thus, we believe that embedded domain-specific language with blockchain can solve most of the problems for multi-agent systems.
- Role-oriented programming with smart contracts is challenging because the choices of technology can affect the result of the study. For instance, stateful and stateless contracts are not advanced technologies that can employ all of the features of the object-oriented paradigm. Turing complete and non-Turing complete technologies will be scrutinized in future work. In a nutshell, this paper presented a new significant role approach with smart contract programming implementing hash data structure and providing data secu- rity regarding roles. Presenting our approach will simplify the application develop- ment process for further researchers.

In particular, this study inspects role-oriented multi-agent systems regarding domain-driven design and blockchain technology in multi-agent systems separately. The course of finding common points between various research areas guides us to design a domain-specific language to contribute to the multi-agent system area. There is no common understanding in terms of multi-agent system methodology, analysis, design, or implementation. This increases the complexity of the research in the multi-agent system area. The limited number of domain-driven agent-oriented languages have been provided so that one can notice that multi-agent system research is most likely conceptual and it does not provide prototype and result-evaluated research. Synthesized metamodeling from scratch in different research areas can be ambiguous; thus, we believe that embedded domain-specific language with blockchain can solve most of the problems for multi-agent systems. Consequently, this paper presented a new significant role approach with smart contract

programming to encrypt data and to provide data security regarding roles. Introducing this approach will simplify the application development process with an embedded domain specific language.

## 7 Future Work

In future work, we will focus on the following features:

- Runtime metaprogramming extendability of the GasMASk code generation.
- Dynamic smart contract integration into the embedded-domain specific language.

We will research on the runtime feature of code generation with a statically typed language, which means that a language can do type checking the type of variable at compile time. Most of the dynamically typed language can handle this; however, most of the agent frameworks are written in a statically typed language. Another consideration would be regarding role types and natural types at runtime adaptability. Separately, we will conduct research on dynamic adaptability on a deterministic blockchain environment.

## Acknowledgements

The author would like to thank his supervisors, Prof. Dr. Uwe Aßmann, and Prof. Dr. Susanne Strahringer, for the patient guidance, encouragement, and comments they have provided to shape his doctoral vision. This work is funded by the German Research (DFG) within the Research Training Group Role-Based Software Infrastructures for continuous-context-sensitive Systems (GRK 1907, TU Dresden, Software Technology Group, Nöthnitzer Straße 46, 01187, Dresden).

## References

1. *Index*. John Wiley and Sons, Ltd, 2004, pp. 221–225. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/0470861223.index
2. B. M. Balachandran, "Developing intelligent agent applications with jade and jess," in *Knowledge-Based Intelligent Information and Engineering Systems*, I. Lovrek, R. J. Howlett, and L. C. Jain, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 236–244.
3. J. A. M. van den Brand A. Serebrenik J.J. Brunekree, *Code Generation with Templates*. John Wiley and Sons, Ltd, 2006, pp. 221–225.
4. S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. USA: Prentice Hall Press, 2009.
5. N. Gaur, L. Desrosiers, P. Novotny, V. Ramakrishna, A. O'Dowd, and S. A. Baset, *Hands-On Blockchain with Hyperledger: Building Decentralized Applications with Hyperledger Fabric and Composer*. Packt Publishing, 2018.
6. N. Taing, T. Springer, N. Cardozo, and A. Schill, "A dynamic instance binding mechanism supporting run-time variability of role-based software systems," 03 2016, pp. 137–142.

7. N. Ubayashi and T. Tamai, "Roleep: Role based evolutionary programming for cooperative mobile agent applications," *Principles of Software Evolution, International Symposium on*, vol. 0, p. 232, Nov. 2000.
8. B. Ferreira and A. Leitão, "Context-oriented algorithmic design," in *SLATE*, 2018.
9. K. Wan and V. Alagar, "A context-aware trust model for service-oriented multi-agent systems," in *Service-Oriented Computing – ICSOC 2008 Workshops*, G. Feuerlicht and W. Lamersdorf, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 221–236.
10. T. Kühn, S. Böhme, S. Götz, and U. Aßmann, "A combined formal model for relational context-dependent roles," in *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering*, ser. SLE 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 113–124. [Online]. Available: https://doi.org/10.1145/2814251.2814255
11. I. Bashir, *Advanced blockchain development : build highly secure, decentralized applications and conduct secure transactions*, 1st ed., ser. Learning path. Birmingham ;: Packt Publishing, 2019.
12. A. Imeri, J. Lamont, N. Agoulmine, and D. Khadraoui, "Model of dynamic smart contract for permissioned blockchains," 11 2019.
13. J. Ferber and O. Gutknecht, "A meta-model for the analysis and design of organizations in multi-agent systems," Feb. 1970.
14. G. Cabri, L. Leonardi, L. Ferrari, and F. Zambonelli, "Role-based software agent interaction models: A survey," *Knowledge Eng. Review*, vol. 25, pp. 397–419, Dec. 2010.
15. G. Cabri, L. Leonardi, and F. Zambonelli, "Implementing role-based interactions for internet agents," Feb. 2003, pp. 380– 387.
16. Cabri, Giacomo and Leonardi, Letizia and Zambonelli, Franco, "Brain: A framework for flexible role-based interactions in multiagent systems," vol. 2888, Nov. 2003, pp. 145–161.
17. A. Shukla, S. K. Mohalik, and R. Badrinath, "Smart contracts for multiagent plan execution in untrusted cyber-physical systems," in *2018 IEEE 25th International Conference on High Performance Computing Workshops (HiPCW)*, 2018, pp. 86–94.
18. U. Kulesza, A. Garcia, C. Lucena, and P. Alencar, "A generative approach for multi-agent system development," vol. 3390, May 2004, pp. 52–69.
19. D. Ghosh, *DSLs in Action*, 1st ed. USA: Manning Publications Co., 2010.


## Author

**Orçun Oruç** received M.Sc. from TU Chemnitz, and he graduated from Kocaeli University with a B.Sc. degree. Currently, he is pursuing his Ph.D. in Computer Engineering-Software Technology at the Dresden Technical University. His research interests include programming languages, multi-agent systems, role-oriented programming, natural language processing, decentralized, and distributed applications.